Date 9/10/21

Neeraj Appam 7073

SDS Page No. Date / /

A.I Practical 11

Aim - Implement passive reinforcement learning algorithm based on adaptive dynamic programming (ADP) for the 3 by 4 word problem

Algorithm.
1) 4 actions, up, left, right, down & function return state utility
2) Starting state vector
3) The agent start from (1,1)
4) Transition matrix loaded from file
T = np.load
5) performing policy evolution
6) perform policy improvement
7) obtain q from u
8) Perform policy iteration
9) provide value iteration
10) Save and load and model
11) plot graph
12) stop

p11 ag.py - E:\fffiiles\college pracs and projects\AI\p11 ag.py (3.8.3)

File Edit Format Run Options Window Help

```python
# Import libraries
import os
import random
import gym
import copy
import pickle
import numpy as np
import matplotlib.pyplot as plt

#Neeraj Appari S073

# Plot values
def plot_values(V):

    # reshape value function
        V_sq = np.reshape(V, (8,8))
        # plot the state-value function
        fig = plt.figure(figsize=(10, 10))
        ax = fig.add_subplot(111)
        im = ax.imshow(V_sq, cmap='RdBu')
        for (j,i),label in np.ndenumerate(V_sq):
            ax.text(i, j, np.round(label, 5), ha='center', v
        plt.tick_params(bottom='off', left='off', labelbott
        plt.title('State-Value Function')
        plt.show()
# Perform a policy evaluation
def policy_evaluation(env, policy, gamma=1, theta=1e-8):
    V = np.zeros(env.nS)
    while True:
        delta = 0
        for s in range(env.nS):
            Vs = 0
```

Python 3.8.3 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19)
[MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for mor
e information.
>>>
========= RESTART: E:\fffiiles\college pracs and projects\
AI\p11 ag.py =========
--- Frozen Lake ---
Observation space: Discrete(64)
Action space: Discrete(4)

--- One-step dynamics
[(0.3333333333333333, 1, 0.0, False), (0.3333333333333333,
0, 0.0, False), (0.3333333333333333, 9, 0.0, False)]

Optimal Policy (LEFT = 0, DOWN = 1, RIGHT = 2, UP = 3):
Squeezed text (64 lines).
```

Type here to search

08:17 12-10-2021

p11 ag.py - E:\fffiiles\college pracs and projects\AI\p11 ag.py (3.8.3)

File Edit Format Run Options Window Help

```python
        im = ax.imshow(V_sq, cmap='RdBu')
        for (j,i),label in np.ndenumerate(V_sq):
            ax.text(i, j, np.round(label, 5), ha='center', va='center', fontsize=12)
        plt.tick_params(bottom='off', left='off', labelbottom='off', labelleft='off')
        plt.title('State-Value Function')
        plt.show()
# Perform a policy evaluation
def policy_evaluation(env, policy, gamma=1, theta=1e-8):
    V = np.zeros(env.nS)
    while True:
        delta = 0
        for s in range(env.nS):
            Vs = 0
            for a, action_prob in enumerate(policy[s]):
                for prob, next_state, reward, done in env.P[s][a]:
                    Vs += action_prob * prob * (reward + gamma * V[next_state])
            delta = max(delta, np.abs(V[s]-Vs))
            V[s] = Vs
        if delta < theta:
            break
    return V
# Perform policy improvement
def policy_improvement(env, V, gamma=1):

        q = q_from_v(env, V, s, gamma)

        # OPTION 1: construct a deterministic policy
        # policy[s][np.argmax(q)] = 1

        # OPTION 2: construct a stochastic policy that puts equal probability on maximizing actions
        best_a = np.argwhere(q==np.max(q)).flatten()
        policy[s] = np.sum([np.eye(env.nA)[i] for i in best_a], axis=0)/len(best_a)
```

Ln: 17 Col: 32

Type here to search

08:17 12-10-2021 ENG

p11 ag.py - E:\fffiiles\college pracs and projects\AI\p11 ag.py (3.8.3)

File Edit Format Run Options Window Help

```
            q[a] += prob * (reward + gamma * V[next_state])
    return q
# Perform policy iteration
def policy_iteration(env, gamma=1, theta=1e-8):
    policy = np.ones([env.nS, env.nA]) / env.nA
    while True:
        V = policy_evaluation(env, policy, gamma, theta)
        new_policy = policy_improvement(env, V)

        # OPTION 1: stop if the policy is unchanged after an improvement step
        if (new_policy == policy).all():
            break;

        # OPTION 2: stop if the value function estimates for successive policies has converged
        # if np.max(abs(policy_evaluation(env, policy) - policy_evaluation(env, new_policy))) < theta*1e2:
        #    break;

        policy = copy.copy(new_policy)
    return policy, V
# Truncated policy evaluation
def truncated_policy_evaluation(env, policy, V, max_it=1, gamma=1):
    num_it=0
    while num_it < max_it:
        for s in range(env.nS):
            v = 0
            q = q_from_v(env, V, s, gamma)
            for a, action_prob in enumerate(policy[s]):
                v += action_prob * q[a]
            V[s] = v
        num_it += 1
    return V
# Truncated policy iteration
```

Ln: 17 Col: 32

Type here to search

08:17
12-10-2021

p11 ag.py - E:\fffiiles\college pracs and projects\AI\p11 ag.py (3.8.3)

File Edit Format Run Options Window Help

```
    with open('models\\frozen_lake' + type + '.adp', 'wb') as fp:
        pickle.dump(bundle, fp)
# Load a model
def load_model(type:str) -> ():
    if(os.path.isfile('models\\frozen_lake' + type + '.adp') == True):
        with open ('models\\frozen_lake' + type + '.adp', 'rb') as fp:
            return pickle.load(fp)
    else:
        return (None, None)
# The main entry point for this module
def main():
    # Create an environment
    env = gym.make('FrozenLake8x8-v1', is_slippery=True)
    # Print information about the problem
    print('--- Frozen Lake ---')
    print('Observation space: {0}'.format(env.observation_space))
    print('Action space: {0}'.format(env.action_space))
    print()
    # Print one-step dynamics (probability, next_state, reward, done)
    print('--- One-step dynamics')
    print(env.P[1][0])
    print()
    # (1) Random policy
    #model, V = load_model('1')
    model = np.ones([env.nS, env.nA]) / env.nA
    V = policy_evaluation(env, model)
    print('Optimal Policy (LEFT = 0, DOWN = 1, RIGHT = 2, UP = 3):')
    print(model,'\n')
    plot_values(V)
    save_model((model, V), '1')
    # (2) Policy iteration
    ##model, V = load_model('2')
```

Ln: 17 Col: 32

Type here to search

ENG 08:17 12-10-2021

p11 ag.py - E:\fffiiles\college pracs and projects\AI\p11 ag.py (3.8.3)

File Edit Format Run Options Window Help

```
    #save_model((model, V), '3')
    # (4) Value iteration
    ##model, V = load_model('4')
    #model, V = value_iteration(env)
    #print('Optimal Policy (LEFT = 0, DOWN = 1, RIGHT = 2, UP = 3):')
    #print(model,'\n')
    #plot_values(V)
    #save_model((model, V), '4')
    # Variables
    episodes = 10
    timesteps = 200
    total_score = 0
    # Loop episodes
    for episode in range(episodes):
        # Start episode and get initial observation
        state = env.reset()
        # Reset score
        score = 0
        # Loop timesteps
        for t in range(timesteps):
            # Get an action (0:Left, 1:Down, 2:Right, 3:Up)
            action = get_action(model, state)

            # Perform a step
            # Observation (position, reward: 0/1, done: True/False, info: Probability)
            state, reward, done, info = env.step(action)
            # Update score
            score += reward
            total_score += reward
            # Check if we are done (game over)
            if done:
                # Render the map
```

Ln: 17 Col: 32

Type here to search

ENG 08:17 12-10-2021

p11 ag.py - E:\fffiiles\college pracs and projects\AI\p11 ag.py (3.8.3)

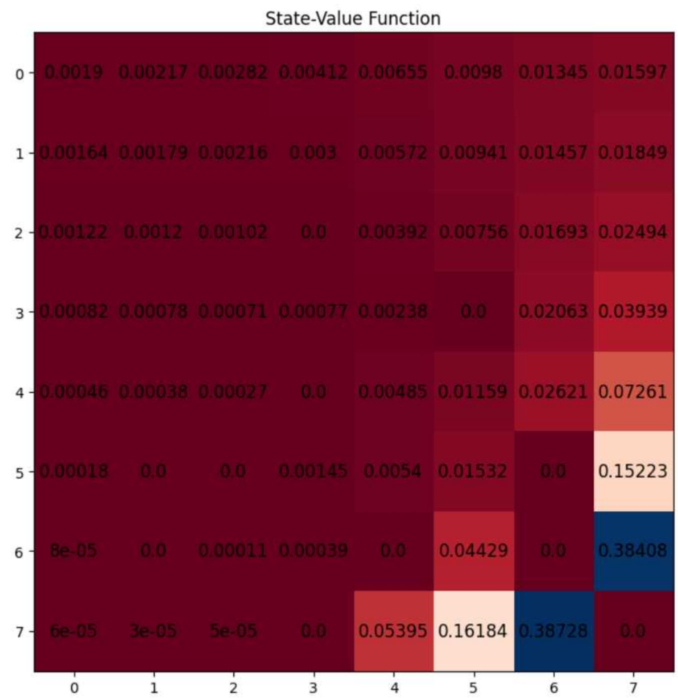
File Edit Format Run Options Window Help

```
        state = env.reset()
        # Reset score
        score = 0
        # Loop timesteps
        for t in range(timesteps):
            # Get an action (0:Left, 1:Down, 2:R
            action = get_action(model, state)

            # Perform a step
            # Observation (position, reward: 0/1
            state, reward, done, info = env.step
            # Update score
            score += reward
            total_score += reward
            # Check if we are done (game over)
            if done:
                # Render the map
                print('--- Episode {} ---'.forma
                env.render(mode='human')
                print('Score: {0}, Timesteps: {1
                print()
                break

    # Close the environment
    env.close()
    # Print the score
    print('--- Evaluation ---')
    print ('Score: {0} / {1}'.format(total_score
    print()
# Tell python to run main method
if __name__ == "__main__": main()
```

Figure 1



Type here to search

ENG 08:17 12-10-2021